A Two-step Optimization Technique for Functions Placement, Partitioning, and Priority Assignment in Distributed Systems

Asma Mehiaoui, Ernest Wozniak
CEA LIST DILS
{asma.mehiaoui, ernest.wozniak}@cea.fr

Sara Tucci-Piergiovanni, Chokri Mraidha
CEA LIST DILS
{sara.tucci, chokri.mraidha}@cea.fr

Marco Di Natale
Scuola Superiore Sant'Anna
marco@sssup.it

Haibo Zeng
McGill University
haibo.zeng@mcgill.ca

Jean-Philippe Babau, Laurent Lemarchand
Lab-STICC, University of Brest
{jean-philippe.babau, laurent.lemarchand}@univ-brest.fr

Abstract

Modern development methodologies from the industry and the academia for complex real-time systems define a stage in which application functions are deployed onto an execution platform. The deployment consists of the placement of functions on a distributed network of nodes, the partitioning of functions in tasks and the scheduling of tasks and messages. None of the existing optimization techniques deal with the three stages of the deployment problem at the same time. In this paper, we present a staged approach towards the efficient deployment of real-time functions based on genetic algorithms and mixed integer linear programming techniques. Application to case studies shows the applicability of the method to industry-size systems and the quality of the obtained solutions when compared to the true optimum for small size examples.

Categories and Subject Descriptors C.3 [*Real-time and embedded systems*]; G.1.6 [*Optimization*]: linear programming; stochastic programming

Keywords Distributed real-time applications, response-time analysis, optimization, linear programming, genetic algorithm, placement, partitioning, scheduling

1. Introduction

In the development of real-time cyber-physical systems, abstraction levels are used to manage complexity [17]. Industrial standards (like the automotive AUTOSAR [2] and the Model-Driven Architecture from the OMG [1]) and academic frameworks (including the Platform-Based Design [27]) recommend system development along the lines of the Y-chart approach [15]: a functional model representing the system functions and the signals exchanged among them is deployed onto an execution platform model consisting of nodes, buses, tasks and messages. End-to-end real-time constraints (deadlines) are specified on transactions, that is, chains of functions, activated by an external event (e.g. as detected by a sensor) or a timer, and terminating with the execution of a sink function (e.g. sending a command to actuators) [2]. Classically as in AUTOSAR, two subsequent and separated activities (often carried out by different teams) are dedicated to functional deployment. First, the *placement* of functions on execution nodes is defined. Next, the *partitioning* of the set of functions and signals in tasks and messages and the *scheduling* of tasks and messages is computed. In fixed-priority scheduled systems, this includes the assignment of priorities. Once the functional level is arranged in tasks and messages, the worst-case timing behavior can be computed and compared against end-to-end deadlines. Ideally, the placement, partitioning, and scheduling problem should be automated by solving an optimization problem with respect to the metrics and real-time constraints, but because of its inherent complexity – placement, partitioning, and scheduling are NP-hard problems – suboptimal staged approaches and heuristics are used. The first stage (as in [17]) may be dedicated to placement in isolation, i.e. functions and signals are assigned to nodes and buses without the definition of the task and message model (including their priorities). As the deployment is only partial, worst-case latencies cannot be evaluated and the placement is evaluated by simple metrics and constraints, such as resource utilization. Other approaches lie in solving the placement and scheduling of tasks and messages with respect to latency-based constraints and metrics [8, 32]. In this case, the function-to-task and signal-to-message partitioning is previously solved in isolation with simple heuristics, such as grouping in one task all functions belonging to the same transaction and/or executing at the same rate or a one-to-one mapping between functions and signals to tasks and messages. Both approaches may lead to sub-optimal deployments.

In this paper we are intersted in tackling function placement, partitioning and scheduling exploring the use of Mixed Integer Linear Programming (MILP) and Genetic Algorithms (GA). MILP and GA algorithms are complementary and may derive mutual benefits from a joint application to the problem. An MILP formulation is easily extensible, re-targetable to a different optimization metric and can easily accommodate additional constraints or legacy components. An MILP formulation may benefit from the application of powerful commercial solvers. The method also *guarantees optimality in case the solver terminates* (when the problem size is manageable) and the distance of the current solution from the optimum can be bounded at any time (computed as the gap between the current best solution and the one of the relaxed linear problem), thus it is possible to evaluate the quality of the intermediate solutions generated by the solver. GA solutions typically scale much better. However, the quality of the solution provided by GA depends on many factors, for example, the appropriate choice of a

LCTES'13, June 20–21, 2013, Seattle, Washington, USA.

crossover operator, and is very hard to evaluate. Using both techniques, the MILP solver can provide the true optimum for average size problems, helping in the tuning of the GA formulation and the assessment of its quality. For larger size problems, it can provide an upper bound to the optimum metric, helping in the evaluation of the quality obtained by the GA.

Our Contributions. In this paper we first present a MILP technique that solves the placement, partitioning and scheduling problem (from now on PPS) at the same time (unlike existing approaches). The MILP formulation of the problem can be solved and returns the optimal solution, but it is practically applicable only to small-size systems. In order to scale to industry-size systems we then propose a staged, divide-and-conquer approach with an iterative improvement optimization. Through divide-and-conquer the PPS is divided in two sub-problems of manageable size solved in cascade. Response-time optimization and latency constraints are considered for both sub-problems. This staged approach has been implemented using MILP and GA techniques. Of course, when a staged MILP solution is used, several benefits are lost, including the guarantee of optimality. However, the staged MILP solution provides a good tradeoff between scalability (a primary concern for industrial size systems) and the quality of solutions. Furthermore the MILP solution is useful to well-design the staged GA solution, which scales to even larger systems.

The paper is organized as follows. Section 2 presents the related work. Section 3 presents basic definitions and assumptions, then formally introduces the PPS problem. Section 4 presents the staged optimization strategy for the PPS problem and the two proposed formulations (MILP and GA) for the inner optimization stages. Section 5 shows the experimental results and Section 6 concludes the paper.

2. Related work

Most automotive controls are designed based on *run-time static priority scheduling* of tasks and messages. Examples of standards supporting this scheduling model are the AUTOSAR operating system [2], and the CAN bus [10] arbitration model.

Optimization techniques have been extensively used to find good solutions to deployment problems. [3] classifies 188 papers along many axes, i.e. design goals and constraints, degrees of freedom, problem solved. None of the surveyed papers, however, considers worst-case latency of the deployed transactions as either design constraint or goal.

End-to-end deadlines have been discussed in research work in single-processor and distributed architectures. In transaction-based activation models (such as the holistic and jitter propagation model in [24, 28] and the transaction model with offsets in [21]), messages are queued by sender tasks and the arrival of messages at the destination node triggers the activation of the receiver task. In such models, task and message schedulers have cross dependencies because of the propagation of the activation signals and real-time analysis can be performed using the holistic model [28] [23] based on the propagation of the release jitter along the computation path. When offsets can be enforced for tasks and messages to synchronize activations, the system can be analyzed as in [21].

Despite advances in timing analysis, the optimized deployment synthesis problem has not received comparable attention. In [24] the authors discuss the use of genetic algorithms for optimizing priority and period assignments to tasks with respect to an extensibility metric and a number of constraints, including end-to-end deadlines and jitter. In [9], the authors describe a procedure for period assignment on priority-scheduled single-processor systems. In [22, 23], a heuristics-based design optimization algorithm for mixed time-triggered and event-triggered systems is proposed. The algorithm, however, assumes that nodes are synchronized. An integrated optimization framework is also proposed in [14] for systems with periodic tasks on a network of processor nodes connected by a time-triggered bus. The framework uses Simulated Annealing (SA) combined with geometric programming to hierarchically explore task allocation, task priority assignment, task period assignment and bus access configuration.

In [8], task allocation and priority assignment were defined with the purpose of optimizing the extensibility with respect to changes in task computation times. The proposed solution was based on simulated annealing. In [13], a generalized definition of extensibility on multiple dimensions (including changes in the execution times of tasks but also period speed-ups and possibly other metrics) was presented. Also, a randomized optimization procedure based on a genetic algorithm was proposed to solve the optimization problem. The focus is on the multi-parameter Pareto optimization, and how to discriminate the set of optimal solutions. Other works assume a communication-by-sampling model, in which tasks and messages are activated periodically and exchange information over shared variables. In this context, [32] provides an MILP formulation for the problem that considers the placement, scheduling and signal partitioning as degrees of freedom. The formulation is extended to consider extensibility in [30]. Azketa et al. used genetic algorithms to assign priorities to tasks and messages, then to map tasks and messages on the execution platform for event-triggered systems [6]. Finally, a mixed model in which information can be exchanged synchronously (with tasks and messages activating the successors, as in the transaction model) or by periodic sampling is considered in [29], where only the optimization of the activation modes is provided (task and message placement and priorities are fixed). A common characteristic of the above cited works is that function placement and partitioning is not considered.

Functions partitioning problem was considered in [7] [26] and [16] but only for a single-processor system, i.e., without the placement problem.

3. System model and the PPS problem

This section introduces the basic definitions and assumptions on the system model. Then, the PPS problem is defined and most significant parts of the MILP formulation are given.

3.1 System model

The considered system consists of a *physical* architecture (Network topology) and a *logical* architecture(Functional graph).

The network topology is represented by a graph $\zeta^{'}$ of execution nodes $C = \{c_1, c_2, ..., c_c\}$ connected through buses $\beta = \{b_1, b_2, ..., b_\beta\}$. Each node runs a real-time operating system with a preemptive fixed-priority scheduling (such as an AUTOSAR OS). Communication buses are assumed to be Controller Area Networks (CAN), arbitrated by priority (the identifier field of the message). The execution nodes and communication buses may have different processing and transmission speeds. Both execution nodes and communication buses have a maximal capacity utilization, respectively μ_c and μ_β, that must not be exceeded. Each execution node offers a set of tasks that perform the computation required by the system functions.. We denote the set of tasks offered by all the execution nodes as $T = \{t_1, t_2, , \ldots, t_n\}$. Each task t_i is characterized by a set of functions that it executes and a priority level π_{t_i}. We denote as τ_{f_i} the task to which f_i belongs. In turn, communication buses offer a set of messages $\psi = \{m_1, m_2, ..., m_m\}$ that realize the transmission of signals between remote tasks. Each message m_i is defined by the set of signals that it transmits and a priority level π_{m_i}.

The functional graph depicts the operational process of the system, in which events generated by sensors or by users trig-

ger the computation of a set of control functions or algorithms, which eventually define a system response. This behavior may be captured by a dataflow model, and described as a directed acyclic graph ζ composed by a set of concurrent linear transactions $\zeta = \{\Gamma_1,\Gamma_2,...,\Gamma_\Gamma\}$. Each transaction is a 2-tuple,{F, ρ}, where F=$\{f_1,f_2,...,f_f\}$ is the set of functions that represent the atomic operations and $\rho = \{l_1,l_2,...,l_l\}$ is a set of links representing the interactions between functions. Functions exchange data through signals on these links. $\Phi = \{s_1,s_2,...,s_s\}$ denotes the set of signals. A transaction Γ_i is triggered by an external event e_i that can be periodic or sporadic with, respectively, an activation period or a minimum inter-arrival time, denoted in both cases as P_i. Functions and signals within a transaction inherit their period (respectively P_{f_i} and P_{s_i}) from the activation period of the external event triggering this transaction. In addition, each transaction Γ_i has a deadline D_i that represents the maximum time value allowed for the associated transaction to be executed. Functions and signals are, respectively, characterized by a vector of worst-case execution times (WCETs) $\overrightarrow{\omega_{f_i}} = (\omega_{f_i,c_1}, \omega_{f_i,c_2}, ..., \omega_{f_i,c_c})$ and worst-case transmission times (WCTTs) $\overrightarrow{\omega_{s_i}} = (\omega_{s_i,b_1}, \omega_{s_i,b_2}, ..., \omega_{s_i,b_\beta})$, where ω_{f_i,c_c} and ω_{s_i,b_β} are respectively the WCET of f_i on node c_c and the WCTT of s_i on bus b_β.

The event-triggered activation model [28] is considered for functions and signals. In this model, the first function in each transaction Γ_i is triggered by an external event e_i. Subsequent functions are activated upon the completion of the predecessor function (if local) or the arrival of the message delivering the data values for its incoming signal (if remote). Messages are transmitted upon the completion of the sender functions. Figure 1 shows an example.

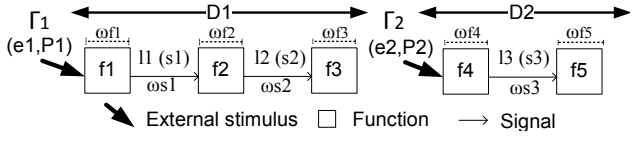


Figure 1. Example of a functional graph

3.2 PPS problem for system response-time optimization

The PPS problem consists in three sub-problems, namely *placement*, *partitioning* and *scheduling*. Placement consists in finding an execution node for each function and a bus for each signal. Partitioning decides which functions/signals to execute/transmit in each task or message. Scheduling is the problem of finding an execution or transmission order among tasks and messages in the same node/bus, this order is expressed by a priority order.

The WCET and WCTT of each function and signal is chosen from its WCETs and WCTTs vector, respectively. This choice depends on placement decisions. Then in the partitioning and scheduling stage, each task gets its set of functions as the group of functions with the same priority level and inherits this priority level. In the same way, a message is constructed such as it contains a set of signals with the same priority level.

Given a functional graph and a network topology, we are interested in finding a *valid* placement, partitioning and scheduling optimized with respect to the system response time. In this paper, we consider two metrics to express the optimization of system response time, the minimization of end-to-end transactions latencies and the maximization of the minimum transactional slack time, as detailed later in this section.

The constraints for the problem are detailed with their associated MILP formulation. For reasons of space availability, we do not report the full MILP formulation but only the most significant or original parts (the full description is available from [18]). For what concerns the variables representing tasks and messages used in MILP formulation, since their number is unknown before optimization, we reserve one task and message slot for each function/signal on each node/bus. Empty slots are not considered in the formulation of the constraints and the metric function. In the following, when clear from the context, task slots and messages slots will be called tasks and messages.

Partitioning constraints include *harmonic rate* and *functional partitioning* constraints. Harmonic rate constraints prevent two functions/signals with non harmonic periods from being mapped into the same task/message. Functional partitioning constraints consists in mapping each function into exactly one task and each signal into at most one message. $A_{i,k}$ is a boolean variable set to 1 if function f_i (signal s_i) is placed on task t_k (message m_k). $X_{i,j,k}$ is also a boolean, and denotes whether both f_i and f_j are partitioned to t_k. The following constraints guarantee that each function is assigned to exactly one task and ensure the consistent definition of $X_{i,j,k}$. Constraints for signal and messages can be found in [18].

$$\sum_{t_k \in T} A_{i,k} = 1$$
$$X_{i,j,k} \leq A_{i,k}, \quad X_{i,j,k} \leq A_{j,k}, \quad X_{i,j,k} + 1 \geq A_{i,k} + A_{j,k}$$

Placement constraints are divided into two subsets. The first concerns *resource utilization* and consists in meeting the maximum capacity utilization of all execution nodes and communication buses. The second relates to *allocation*, and includes: (i) fixed allocation, when a function must be allocated to a specific node (e.g. a function responsible for collecting data from sensors has to be placed on the node linked to the sensor); (ii) exclusive allocation enforcing the placement of each function on exactly one node and each signal on one bus at most; and (iii) bus allocation enforcing the mapping of a signal on the bus connecting the nodes on which its sender and receiver functions are executed. This last constraint needs to take into account the topology of the communication network (e.g. if c_1 and c_2 are not connected then communicating functions f_1 and f_2 cannot be placed on c_1 and c_2, respectively). Function f_i is placed on node c_j if and only if it is placed on task t_k allocated to node c_j. The parameter $TN[j]$ represents the set of tasks slots for c_j. $An_{i,j}$ is a boolean variable which denotes whether f_i is placed on c_j. $X2_{i,j,k}$ is another boolean variable, with value 1 if f_i and f_j are placed on c_k and 0 otherwise. The following constraints ensure the consistency of the definitions of $X2_{i,j,k}$.

$$An_{i,j} = \sum_{t_k \in TN[j]} A_{i,k}$$
$$X2_{i,j,k} \leq An_{i,k}, X2_{i,j,k} \leq An_{j,k}, X2_{i,j,k} + 1 \geq An_{i,k} + An_{j,k}$$

Signal s_i is placed on bus b_j if and only if it is placed on the message slot m_k belonging to b_j. $MB[j]$ represents the set of messages slots belonging to b_j. $Ab_{i,j}$ is a boolean variable which denotes whether the signal s_i is transmitted over the bus b_j.

$$Ab_{i,j} = \sum_{m_k \in MB[j]} A_{i,k}$$

G_i is a boolean variable equal to 1 if s_i is transmitted on a bus and 0 if s_i is a local signal; $NB[j]$ is the set of nodes communicating through bus b_j. The following constraints enforce the conditions for s_i to be on b_j.

$$Ab_{i,j} \leq G_i,$$
$$0 \leq \sum_{c_k \in NB[j]} An_{rec_i,k} + G_i + \sum_{c_k \in NB[j]} An_{snd_i,k} - 3 \cdot Ab_{i,j} \leq 2$$

where snd_i and rec_i are, respectively, the source and destination function of the signal s_i.

Execution order constraints are *local total order* and *functional order* constraints. The local total order constraints consist

in a total priority order for tasks/messages belonging to the same node/bus. However, within each node and bus we assign a different priority order to task t_i and message m_j slots, which is denoted respectively as π_i and π_j. Functional order constraints represent partial orders of execution (such as t_j may not be executed before t_i when t_j depends on the output of t_i). Tasks dependencies are derived from the dependencies of the functions mapped in them. The (constant) parameter $dp_{i,j} = 1$ indicates that the execution of f_j depends on f_i in the functional graph. In this case, f_j cannot be placed on a task with priority higher than the task slot on which f_i is placed.

$$\forall f_i, f_j \ \ s.t. \ dp_{i,j} = 1 \ : \ \sum_{k \in T} A_{j,k} \cdot \pi_k \leq \sum_{k \in T} A_{i,k} \cdot \pi_k$$

Response time constraints are common to the placement, partitioning and scheduling sub-problems. Each transaction $\Gamma_i \in \zeta$ has a latency L_i equal to the worst-case response time (WCRT) R_{f_k} of its last function f_k and the latency of each transaction should not exceeds its deadline ($L_i \leq D_i$). To compute the WCRT of functions, we adapt the response time analysis with jitter propagation (as applied to tasks) given in [28] and [21].

The WCRT R_{f_i} of a function f_i is computed by considering all the q function instances (distinct executions of f_i after activation) in the busy period, as follows:

$$R_{f_i} = \max_{q=1,2,\ldots} [W_{f_i}^{(q)} - (q-1)P_{f_i} + J_{f_i}] \tag{1}$$

Since f_i is executed by a task, its release jitter J_{f_i} is the task release jitter, that is, the largest among all the latest release times for the functions in the same task, which is zero if the function has no predecessor (or a predecessor within the same task), or the worst case response time of the signal it receives from a remote predecessor function (s_i is the signal received by f_i.)

$$J_{f_i} = \max_{\tau_{f_i} = \tau_{f_j}, \forall f_j \in F} \{0, R_{s_i}\} \tag{2}$$

$W_{f_i}^{(q)}$ is the completion time of the q^{th} instance of function f_i

$$W_{f_i}^{(q)} = q \cdot (\omega_{f_i,c_k} + \sum_{f_j : \tau_{f_i} = \tau_{f_j}} \omega_{f_j,c_k}) + \sum_{f_j \in hp(f_i)} \left\lceil \frac{W_{f_i}{}^{(q)} + J_{f_j}}{P_{f_j}} \right\rceil \omega_{f_j,c_k} \tag{3}$$

where $hp(f_i)$ refers to the set of higher priority functions than f_i executing on the same node and within different tasks. The last term represents the preemption time from functions belonging to $hp(f_i)$. The completion time is computed for $q = 1, 2, \cdots$ until the busy period ends, that is, an instance completes at or before the activation of the next instance.

When a (higher priority) task contains functions with different periods, its interference is computed as the sum of the interferences of its functions, as shown in Figure 2, in which a task consists of two functions f_1, activated once every two task executions, and f_2 executed every time. The exact response time formula (3) is not suitable for an MILP formulation. It depends on the number of function activations q in the busy period, which is not known a-priori. Hence, we use a necessary-only (optimistic) conditions for feasibility that only considers the response time of the first instance ($q = 1$). The MILP formulation of (3) with $q = 1$ becomes (the integer variables $I_{j,i}$ represent the number of times f_j may interfere with f_i in the worst case)

$$\begin{aligned} &W_{f_i} = \omega_{f_i,c_k} + \sum_{f_j : \tau_i = \tau_j} \omega_{f_j,c_k} + \sum_{f_j \in hp(i)} I_{j,i} \cdot \omega_{f_j,c_k} \\ &I_{j,i} \cdot P_{f_j} - P_{f_j} < W_{f_i} + J_{f_j} \leq I_{j,i} \cdot P_{f_j} \end{aligned} \tag{4}$$

The function response time is equal to its first instance (as computed by the above formula) in case of restricted deadline $L_i \leq D_i \leq P_i$. When end-to-end deadlines may be larger than the periods or the interarrival times of activation events, the formulation using the constraint (4) may compute an optimal solution that is not feasible. This is why all the solutions obtained from the MILP solver must be verified afterwards using the exact response time formula.

The WCRT R_{s_i} of a signal s_i is computed only for signals representing remote communications (otherwise it is equal to zero), using a similar formula as the WCRT of functions, except for an additional term B_{s_i} that represents the blocking time due to the impossibility of preempting messages. The release jitter of a remote signal is the worst-case response time of its sender function.

Constraint (5) represents the computation of B_{s_i} which is the largest WCTT of any message that shares the same communication bus [12] (or even simpler, the transmission time of the largest CAN message). Note that s_i, s_j and s_l are all transmitted on the bus b_k and m_{s_i} represents the message on which s_i is partitioned.

$$\forall s_j \in \Phi : m_{s_i} \neq m_{s_j}, B_{s_i} \geq \omega_{s_j,b_k} + \sum_{\substack{s_l : m_{s_l} = m_{s_j} \\ m_{s_l} \neq m_{s_i}}} \omega_{s_l,b_k} \tag{5}$$

Optimization metrics can be defined based on the system requirements. In this work we tried two formulations: (i) the minimization of the sum of all (or some) transactions latencies, which is a loose indication of the system performance, and (ii) the maximization of the minimum transactional slack time. A slack time for a given transaction is defined as the difference between the deadline and latency of the transaction. Maximizing the minimal transactional slack time (over all transactional slack times) means maximizing the minimum distance between the latency and deadline of the selected transaction (MinSlack = $\min_{\Gamma_i \in \zeta}[D_i - L_i]$). This latter metric can be easily related to the concept of robustness (or extensibility) of the system against changes in the time parameters of some functions.

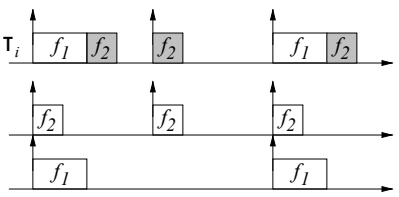


Figure 2. Interference of a task with functions with different harmonic periods as the sum of the function interferences

4. Two-step optimization for PPS

The (one-step) MILP formulation of the PPS problem can only be solved for small systems. To handle industry-size systems, the problem is divided in two smaller (sub)problems and solved in cascade. Two MILP formulations are then provided, one for each sub-problem. A GA counterpart for each sub-problem is also given.

4.1 Overview

Our heuristic (algorithm in Figure 3) combines two classical optimization strategies: divide-and-conquer and iterative improvement.

Divide-and-conquer consists in dividing the PPS problem in two sub-problems solved in cascade, in which placement is solved first (placement problem or PP), and then partitioning and scheduling (PS).

Iterative improvement is used to move towards the optimum. Our algorithm considers an iterative improvement at two levels: inner and outer loops. The inner loop tries to find an optimal system

configuration by applying iteratively an optimization sequence until convergence (two successive solutions are the same). The inner loop consists of the two stages of placement optimization (PP), followed by partitioning and scheduling optimization (PS). In both sub-problems, we aim at optimizing the latency- or slack-based metric. Each iteration starts from a PP step with an initial PS configuration. The PP step provides a new placement of functions/signals to nodes/buses. During this stage, tasks and messages are placed on nodes/buses and so are the functions mapped onto them. Next, the PS step tries to find a new partitioning and scheduling solution that improves the solution found in the PP stage. Depending on the selection of the initial configuration, the PP+PS solution may be a local optimum. To move away from local minima, the outer loop selects set of random initial configurations for partitioning and scheduling as possible starting points.

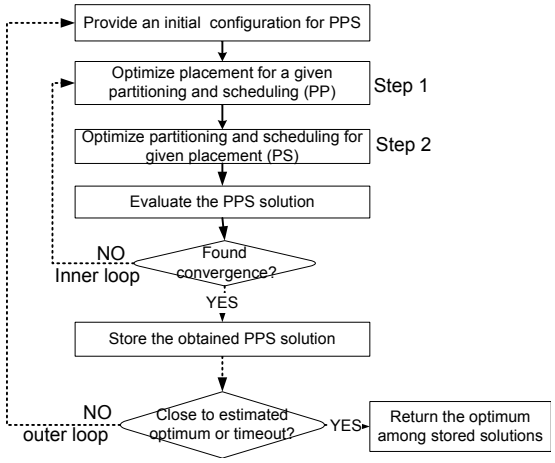


Figure 3. Overview of the Two-step optimization approach

4.2 The MILP formulation for the PP and PS stages

This section outlines the MILP formulation for each sub-problem in the two-step approach.

The two-step formulation uses the same optmization metric for the one-step formulation (Section 3.2).

4.2.1 MILP formulation of the Placement Problem

The objective of this step is optimizing the placement of either an initial configuration or the output of the previous iteration. The parameters are the partitioning of functions (signals) on tasks (messages) and the priorities assigned to tasks and messages. This objective is the optimal placement of tasks/messages and the function/signals in them.

Placement constraints: as mentioned before, they concern allocation and resource utilization constraints. $C[i]$ in (6) is the set of nodes on which t_i is allowed to execute (this includes the case of fixed allocation). $At_{i,c}$ is a binary variable set to 1 if a t_i is placed on node c. Each task must be placed on one node.

$$\Sigma_{c\in C[i]} At_{i,c} = 1 \tag{6}$$

The binary variable $H_{i,j,c}$ based on $At_{i,c}$. $H_{i,j,c}$ denotes whether t_i and t_j are placed on the same node.

$$0 \leq At_{i,c} + At_{j,c} - 2 \cdot H_{i,j,c} \leq 1 \tag{7}$$

Messages placement is based on the placement of tasks. A message can be placed on one bus or to no bus (in case of local communication). The binary variable g_i indicates if m_i is placed on a bus ($g_i = 1$) or is local ($g_i = 0$). The binary variable $Am_{i,j}$ is 1 if m_i is transmitted on bus b_j, 0 otherwise.

$$\Sigma_{b_j\in\beta} Am_{i,j} = g_i \tag{8}$$

m_i is placed on a bus ($g_i = 1$) iff its sender ($sen_{[i]}$) and receiver ($rec_{[i]}$) tasks are on different nodes.

$$\forall(t_i \in sen_{[i]}, t_j \in rec_{[i]}) : 1 - \Sigma_{c\in C} H_{i,j,c} = g_i \tag{9}$$

Since all communicating tasks residing on different nodes must have a connecting bus, the placement of messages must also take into account the network topology. The parameter $NB[j]$ is the set of nodes communicating through the bus b_j. A message m_i is placed on b_j iff the execution nodes of its sender and receiver tasks communicate through b_j.

$$\forall(t_i \in sen_{[i]}, t_j \in rec_{[i]}) :$$
$$0 \leq g_i + \Sigma_{c\in NB[j]} At_{i,c} + \Sigma_{c\in NB[j]} At_{j,c} - 3 \cdot Am_{i,j} \leq 2 \tag{10}$$

$H_{i,j,k}$ indicates if m_i and m_j are transmitted on the same bus b_k. The maximum utilization of nodes is enforced by constraint (11). The utilization of buses is constrained in a similar way.

$$\forall c_j \in C : \Sigma_i A_{i,j} \cdot \left(\omega_{i,c_j}/P_i\right) \leq \mu_j \tag{11}$$

Latency constraints

The task WCRT is the sum of its completion time and jitter.

$$R_i = W_i + J_i \tag{12}$$

The Big M method is used in (15) and (16) to linearize the computation of $I_{i,j,c}$, the maximum number of times that t_j preempts t_i as $I_{i,j,c} = H_{i,j,c} * \sigma_{i,j}$, where $\sigma_{i,j} = \left\lceil \frac{W_i+J_j}{P_j} \right\rceil$. The binary parameter $gp_{i,j}$ denotes whether π_j is higher than π_i.

$$W_i = \sum_{c_p\in C} At_{i,p} \cdot \omega_{i,p} + \sum_{j\in T} \sum_{c_p\in C} I_{i,j,p} \cdot \omega_{j,p} \cdot gp_{i,j} \tag{13}$$

$$0 \leq \sigma_{i,j} - \left(\frac{W_i + J_j}{P_j}\right) < 1 \tag{14}$$

$$\sigma_{i,j} - M * (1 - H_{i,j,c}) \leq I_{i,j,c} \tag{15}$$

$$I_{i,j,c} \leq \sigma_{i,j} \; ; \; I_{i,j,c} \leq M * H_{i,j,c} \tag{16}$$

The jitter of a task is 0 if all its functions are initial and otherwise is equal to the largest WCRT of the received messages (17).

$$\forall m_i \in \psi \;\; s.t \;\; t_i \in rec_{[m_i]} : J_i \geq R_{m_i} \tag{17}$$

The constraints computing the WCRT of messages are similar to (12), (13), (14), (15), and (16) except for the addition of the blocking time B_{m_i} to the formula (13)

$$\forall m_j \in \psi : B_{m_i} \geq \Sigma_{b\in\beta} H_{m_i,m_j,b} \cdot \omega_{m_j,b} \tag{18}$$

Note that the WCRT of a message transmitted inside a node returns only its jitter as its completion time is zero. Thus the task jitter in this case is simply the WCRT of its predecessor.

The result of this MILP formulation consists in a placement of tasks and messages and the input for the next stage is the placement of functions and signals as partitioned in the tasks and messages.

4.2.2 MILP formulation for the Partitioning and Scheduling

At this stage, the placement of functions and signals is given and the following MILP formulation aims at improving (if possible) the partitioning and priority assignment. The MILP formulation assigns priorities to functions and signals within each node and bus, then tasks and messages are constructed based on priorities, i.e. a task (message) is the set of functions (signals) with the same priority order residing on the same node (bus).

Partitioning and scheduling constraints: most constraints are expressed in the same way for functions and signals. In the following, we address only functions and discuss the differences for signals. The binary variable $\chi_{i,j}$ indicates whether f_i has higher priority than f_j. If $\chi_{i,j} = 0$ and $\chi_{j,i} = 0$ then f_i and f_j have the

same priority order, otherwise, either f_i has higher priority than f_j ($\chi_{i,j} = 1$) or f_j has higher priority than f_i ($\chi_{j,i} = 1$).

$$\chi_{i,j} + \chi_{j,i} \leq 1 \tag{19}$$

A set of constraints is used to enforce the symmetric, transitive and inversion properties of the priority order relation and to ensure that each function is partitioned on exactly one task (omitted for space constraints).

A binary variable SP_{f_i,f_j} denotes whether f_i and f_j have the same priority order i.e. they are on the same task.

$$1 = SP_{f_i,f_j} + \chi_{j,i} + \chi_{i,j} \tag{20}$$

Any pair of functions with non-harmonic rates must be assigned to different tasks (have different priorities).

$$\begin{aligned} &\forall f_i, f_j \in F \; s.t. \; f_i \neq f_j : 1 = \chi_{i,j} + \chi_{j,i} \\ &\text{if } P_{f_i} \geq P_{f_j} \text{ and } P_{f_i} \text{modulo } P_{f_j} \neq 0 \end{aligned} \tag{21}$$

The second part of run-time constraints applies only to signals: s_i and s_j cannot be partitioned on the same task if they belong to the same transaction ($\Gamma_{[s_i]} = \Gamma_{[s_j]}$).

$$SP_{s_j,s_i} \neq 0 \text{ if } \Gamma_{[s_i]} = \Gamma_{[s_j]} \tag{22}$$

In constraint (23) the parameter $dp_{f_i,f_j} = 1$ indicates that the execution of f_i depends on f_j in the functional graph. In this case, f_i cannot be placed on a task with priority higher than the task on which f_j is placed.

$$\chi_{i,j} = 0 \text{ if } dp_{f_i,f_j} = 1 \tag{23}$$

Latency constraints: constraints on latency are similar to the previous ones, except that, since tasks are variables in this stage, we compute the WCRT on functions while considering their partitioning. The WCRT of a function f_i is the WCRT of the task containing f_i. Constraint (24) expresses the computation of functions WCRT. Signals WCRT are computed in the same way.

$$R_{f_i} = W_{f_i} + J_{f_i} \tag{24}$$

The first term of the functions completion time (25) represents the WCET of the task containing f_i, as the sum of the WCET of its functions. The binary parameter SN_{f_i,f_j} indicates whether f_i and f_j are placed on the same node. The second term refers to the interferences from higher priority functions f_k that are on the same node as f_i. I_{f_i,f_k}=$\chi_{k,i} \cdot \sigma_{f_i,f_k}$ is a real variable representing the maximum number of times that f_k preempts f_i. σ_{f_i,f_k} is computed as in (14) and I_{f_i,f_k} is linearized using the Big M method.

$$\begin{aligned} W_{f_i} = &\Sigma_{f_j \in F} SP_{f_i,f_j} \cdot \omega_{f_j,c} \cdot SN_{f_i,f_j} \\ &+ \Sigma_{f_k \in F} I_{f_i,f_k} \cdot \omega_{f_k,c} \cdot SN_{f_i,f_k} \end{aligned} \tag{25}$$

The response time of signals is computed similarly, adding a term corresponding to the blocking time B_{s_i} as in (26), where SB_{s_i,s_j} is a binary parameter indicating whether s_i and s_j are transmitted over the same bus. W_{s_i} is 0 if s_i is local.

$$\begin{aligned} \forall s_j \in \Phi : B_{s_i} \geq &\omega_{s_j,b} \cdot (1 - SP_{s_i,s_j}) \cdot SB_{s_i,s_j} \\ &+ \Sigma_{s_k \in \Phi : k \neq j} \omega_{s_k,b} \cdot SP_{s_j,s_k} \cdot SB_{s_j,s_k} \end{aligned} \tag{26}$$

To compute functions jitter we need to define a real variable V_{f_i}, which is 0 if f_i is the first function in its transaction, otherwise, it is the WCRT of the signal it receives.

$$V_{f_i} = \begin{cases} 0 & f_i \text{ is triggered by an external event} \\ R_{s_i} & s_i \text{ is the signal received by } f_i \end{cases} \tag{27}$$

J_{f_i} is computed as the largest V_{f_j} value among all functions with the same priority and belonging to the same node. As above, the Big M method is used to linearize $V_{f_j} \cdot SP_{f_i,f_j}$.

$$\forall f_j \in F \; s.t \; SN_{f_i,f_j} = 1 : J_{f_i} \geq V_{f_j} \cdot SP_{f_i,f_j} \tag{28}$$

Similarly, we use an auxiliary real variable (X_{s_i}) to compute the jitter of signals, which depends on the placement and partitioning of its sender and receiver functions. In (29), the real variables Y_{f_i,f_j} and Z_{f_i,f_j} are equal respectively to $J_{f_i} \cdot SP_{f_i,f_j}$ and $R_{f_i} \cdot (1 - SP_{f_i,f_j})$. These latter variables are also linearized using the Big M method. According to (29) and the above definition of Y_{f_i,f_j} and Z_{f_i,f_j}, X_{s_i} may have two values depending on the variable SP_{f_i,f_j} and the parameter SN_{f_i,f_j}. When the sender (sen_{s_i}) and receiver (rec_{s_i}) functions of s_i are on different nodes, or on the same node but different tasks, X_{s_i}=R_{f_i}. X_{s_i}=J_{f_i} if the sender and receiver functions are on the same task.

$$\begin{aligned} X_{s_i} = (Y_{f_i,f_j} + Z_{f_i,f_j}) \cdot SN_{f_i,f_j} + R_{f_i} \cdot (1 - SN_{f_i,f_j}) \\ s.t \; sen_{s_i} = f_i \; and \; rec_{s_i} = f_j \end{aligned} \tag{29}$$

Since a message may transmit multiple signals, the jitter of a signal should take into account the jitter of all signals transmitted on the same message. Constraint (30) defines J_{s_i} as the largest value of X_{s_j} among all signals belonging to the same message. If s_i is local, then its jitter is simply equal to X_{s_i}.

$$\forall s_j \in \Phi \; s.t \; SB_{s_i,s_j} = 1 : J_{s_i} \geq X_{s_j} \cdot SP_{s_i,s_j} \tag{30}$$

This MILP formulation returns as output a set of functions (signal) partitions as well as a priority order for each partition (task or message) on each node (bus). As priority orders are given locally to a node (bus), a post-processing, which consists in defining a global priority order, is required before the next iteration. The global priority order is simply defined based on the order considered as input in the previous step of the same iteration.

4.3 The GA formulation for the two-step approach

Genetic Algorithm (GA) is an optimization technique patterned after natural selection in biological evolution (Algorithm 4.3.1). In a GA, the space of all possible solutions (feasible and not feasible) to the optimization problem is encoded using a string of bits, called chromosome. Each bit or group of bits in the sequence typically encodes one parameter of the solution (such as the placement of a function or the priority of a task). Several solutions are generated at each round (population), starting from an initial set and then obtaining new solutions by a composition function (or *crossover*) that applies to two chromosomes and produces a new one or by a mutation operator that changes the bit string of a chromosome to generate a new one. Each new generation (or offspring) is evaluated. Some bit strings correspond to non-feasible solutions, or dead individuals and are discarded. A set of the most promising ones is retained and used for computing the next generation.

In the context of our problem, we implement two GA algorithms for the PP and PS stages respectively. GA is used because of its better scalability, in comparison to MILP. Let us note that the quality of the solutions obtained from GA is difficult to evaluate and guarantee, since it is based on many factors, such as the choice of the encoding, crossover and mutation operators, the fitness function and additional mechanisms that improve the search and guarantee constraints.

4.3.1 A GA Solution to the Placement Problem

The *Encoding* definition translates a solution configuration in a string of bits. In the placement problem, a specific solution, i.e. single chromosome, represents the allocation of tasks onto the processing units, and messages on buses. We used the value encoding, in which each gene (subset of bits) in a chromosome contains a specific value. In our case, a gene relates either to a task or a message. The value held by a gene represents an execution node or a communication bus. Although formally PP stage refers to the functions and signals placement here we are encoding tasks and messages. This is due to the knowledge about the functions and signals

partitioning during the PP stage. Therefore the placement of functions and signals can be directly inferred from the tasks/messages allocation. There are two advantages of this encoding. First, the number of tasks/messages is never greater than the number of functions/signals. Hence the size of a chromosome will not be greater if functions/signals were encoded. This can significantly save memory which is an issue especially if large initial populations are considered. Secondly, in case of functions/signals encoding, additional mechanism to preserve the correctness of the chromosome in regards to the allocation constraints would be required. Namely, all the functions/signals of the same task/message have to be allocated on the same processor/bus.

Algorithm 1 General form of a GA Algorithm

```
1: // Define encoding, crossover and mutation operator, fitness
2: // Specify the size of an initial population - P_size
3:
4: Generate initial population
5: while termination condition is not met do
6:     Evaluate each solution from the population P
7:     Generate new population P by applying the crossover and mutation operators
8: end while
9: return the best solution from P
```

Figure 4 shows an example of a chromosome that corresponds to the specific configuration for placement. Gene t_1 holds a value equal to 1, which is the index of the node on which t_1 and all its functions, i.e. f_1 and f_2 will run. The value of the gene m_1 is 0, indicating that message m_1 in this placement configuration is locally transmitted. The selection of the *Crossover* operator is very important for the quality of the GA solution. The operator combines information from two parent chromosomes to create a new child. The choice of the parent chromosomes can be done in many ways, but it is always highly dependent on a chromosome fitness rate. For our implementation we selected the OX3 crossover operator [11] with a tournament selector [19] (with size equal to 5). The OX3 creates two child chromosomes from two parents. It selects two random positions in a chromosome. The values between these points are copied from the first/second parent to the second/first child in the same absolute position. The remaining values are copied from the first/second parent to the first/second child. A simple result of the application of this operator on two chromosomes is shown on the Figure 5. The *Mutation* operator chooses a random point in a chromosome and changes the value of the gene at the selected point to a new random value. If the randomly selected gene corresponds to a task, the new value is chosen from the list of available execution nodes. If it relates to a message, the new value is chosen from the list of available buses.

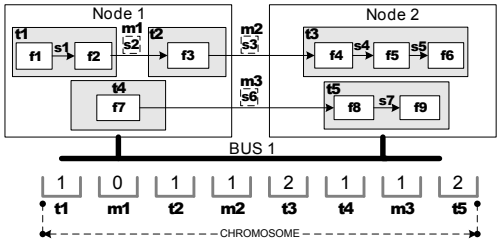


Figure 4. Chromosome for the specific placement configuration

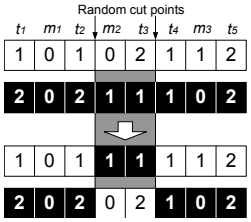


Figure 5. Example application of the crossover

The *Fitness function* defines how much the solution optimizes the performance criteria. Chromosomes are ranked according to this function and, the higher the rank, the higher the probability that the chromosome is selected as a parent for a crossover or the target of a mutation. Our fitness function corresponds to the optimization metrics, that is, in the case of the minimum transactional slack time, it computes $\min_{\Gamma_i \in \zeta}(D_i - L_i)$ where L_i is the latency of the transaction Γ_i and D_i its deadline.

The *Initial population* is generated randomly, i.e. for each task gene, a random number representing its execution node is assigned. However, the initial population does not contain solutions which violate the utilization constraints. Therefore if a generated chromosome leads to the violation of a utilization constraint, we call a correction procedure.

Correction mechanisms are used to avoid the generation of non-feasible solutions in the initial population or after the crossover and mutation. In the case of the violation of utilization constraint, the chromosome is modified by lowering the load of the node(s) with excessive utilization. The procedure randomly selects a task from one of these nodes and then moves it to a destination node, randomly selected among those that can accommodate the additional load. Tasks are moved until a feasible load distribution is found. Incorrect definitions of the communication are also fixed. If two communicating tasks are placed on different nodes, the gene in a chromosome that relates to the message exchanged between the tasks must have a number associated with one of the buses that connect the two nodes. Our correction mechanism checks all message values. Each time an incorrect bus is found, the procedure randomly generates a new bus identifier among those that are valid with respect to the tasks placement.

4.3.2 The GA Formulation for Partitioning and Scheduling

After the definition of the function and signal placement (implicitly by the placement of tasks and messages), the maximum number of new possible tasks and messages for each node and bus can be computed as the number of functions or signals allocated on the resource. Also, signals that result in local communications are not represented in chromosomes. For the PS stage, we only describe the encoding, the generation of the initial population, and the correction mechanism. The crossover mutation operators follow the same logic as in the placement stage. The fitness function is the same.

In the PS *Encoding* each gene represents a function or a signal exchanged among CPUs. The value of the gene is the index of the task or message executing the function or transmitting the signal. The index of a task or message also represents its priority, and its period is the gcd of the functions/signals mapped onto it. In the case of Figure 4, the system partitioning and scheduling is represented by the chromosome shown in Figure 6, where f_1 is executed by t_1 (with priority 1), together with function f_2.

Figure 6. Chromosome for the partitioning and scheduling configuration

The *Initial population* is randomly generated by assigning a task or message index to each function and signal.

The *Correction* function, called when a new chromosome is generated as part of the initial population or after the crossover and

mutation enforces the order of execution constraints. The range of values for a gene is constrained by the values assigned to other genes. If function f_1 precedes f_2, and the gene representing f_1 is assigned to a task with priority π_i, then f_2 should be partitioned on the same task or a task with priority lower than π_i.

5. Experiments

The objective of the following experiments is to assess (i) the quality of solutions obtained with the two-step technique (MILP and GA) against optimal solutions given by the one-step MILP, (ii) comparison of the two-step MILP versus the two-step GA when applied to an industry-size system and (iii) scalability and runtime evaluation of all the techniques.

5.1 Quality evaluation of two-step techniques against one-step MILP

This section presents a first set of small-size tests with the same initial configuration. Results show the convergence of the inner loop and the impact of the initial configuration on the quality of results. The second experiment applies the two-step and the one-step techniques to an automotive case-study. For this case three different initial configurations are tested.

5.1.1 Small-size tests

To evaluate the quality of solutions given by the two-step techniques, we choose six randomly generated systems with functional graphs as in Figure 7). In all these systems, the WCETs of functions and the WCTTs of signals are the same for all nodes and buses, and the network topology consists of two nodes and a single bus. The maximal capacity utilization is set to 1 for the bus and both nodes. The initial configuration for partitioning and scheduling is as follows: (i) each function executes in one task and each signal is transmitted by one message; (ii) priorities are assigned to tasks as follows: (1) if they belong to the same transaction, if the function f_j depends on the function f_i then t_i is higher priority than t_j, (2) between transactions, we follow the deadline-monotonic (DM) approach [5]. Messages inherit the priority of the sending task. Figure 8 shows the comparison of the optimal solutions with the

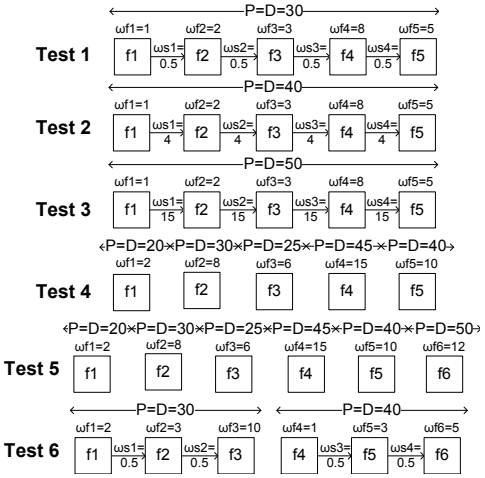
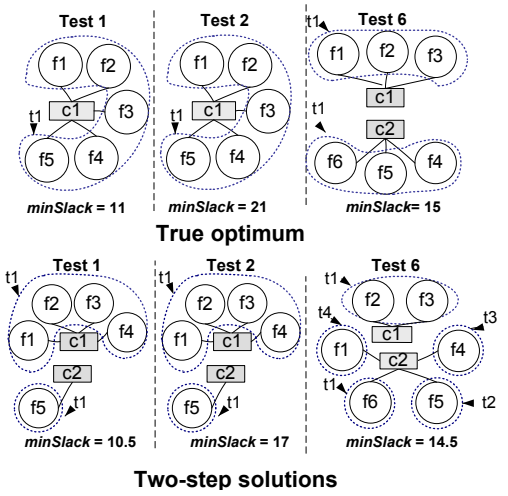
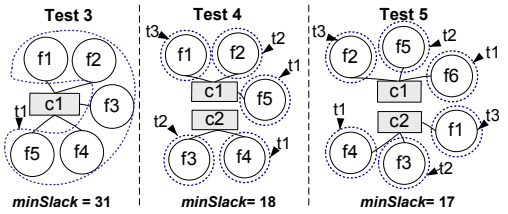


Figure 7. Tests for evaluating solutions quality

solutions provided by the two-step techniques (placement is represented by full lines, partitions are represented by dashed lines and the higher index of tasks represents the higher priority). For these tests the two-step techniques for both MILP and GA computed the same results. For tests 3, 4 and 5 the two-step techniques (MILP and GA) computes the optimal solution Figure 8(b). For tests 1, 2 and 6 they compute a local optimum Figure 8(a). In the three last cases, the selected initial configuration prevents finding the optimum. The two-step techniques return the solutions for which interferences between tasks are minimal. This results in splitting functions between nodes, which in next iterations prevents finding the solution in which all functions are on the same task and node. This situation does not occur for the test 3 where WCTTs are very large with respect to tasks interferences.



(a) Different solutions



(b) Identical solutions

Figure 8. Comparison results between two-step and optimal solutions for tests 1-6

Table 1 shows partial results obtained at each step of the inner loop. For these tests, two-step MILP and GA computed the same results at each step of all iterations. The values in the cells represent the maximum of the minimum transactional slack time (O_s) and the minimum sum of latencies (O_l). At each step a better or equal solution is found. Convergence is obtained in the worst case at the third iteration.

5.1.2 Automotive case study

We consider an automotive system composed of the CCS and ABS sub-systems [4][20]. Figure 9 represents the functional graph of this automotive system. Each function (signal) has the same WCET (WCTT) for all nodes (buses). The network topology for this test is composed of four nodes and a single bus. The maximal capacity utilization is 1 for the bus and all nodes. For this system, we define three initial configurations for partitioning and scheduling as shown in Figure 10. Partitions are represented by dashed lines and

Table 1. Intermediate results for two-step solutions

Test	Steps	Iter 1 O_s/O_l	Iter 2 O_s/O_l	Iter 3 O_s/O_l	Test	Steps	Iter 1 O_s/O_l	Iter 2 O_s/O_l
1	PP	1.5/28	7.5/22.5	10.5/19.5	4	PP	18/59	18/59
	PS	5.5/24	10.5/19.5	10.5/19.5		PS	18/59	18/59
2	PP	7/33	17/23	-/-	5	PP	17/91	17/91
	PS	17/23	17/23	-/-		PS	17/91	17/91
3	PP	7/43	31/19	-/-	6	PP	11.5/36	14.5/24
	PS	31/19	31/19	-/-		PS	14.5/24	14.5/24

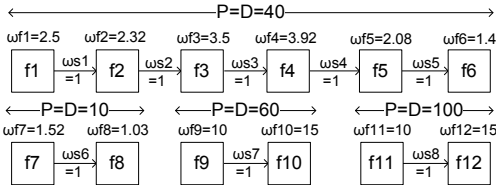


Figure 9. CCS and ABS sub-systems

the index of the partition defines its priority (higher index means higher priority). The optimal solution given by the one-step MILP

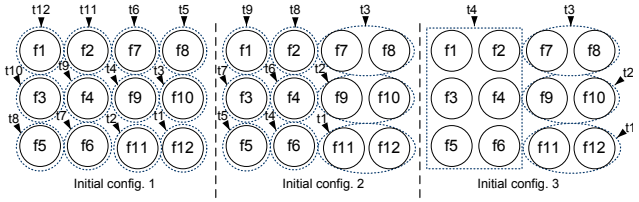


Figure 10. Initial configurations for partitioning and scheduling

is shown in Figure 11 (right-most solution). The values for the minimal slack and latency are respectively 7.45 and 68.27. Table 2 provides the results obtained with the two-step MILP at each iteration of the inner loop for the three selected initial configurations. Cell values refer to the maximum of the minimum transaction slack time (O_s) and the minimum sum of latencies (O_l). The obtained configurations are shown in Figure 11 for the MILP and in Figure 12 for the GA. GA computed different solutions with the same metric value and number of iterations for the first and second configurations. For the third configuration, the same solution was computed by MILP and GA. The optimal solution for this system is the one obtained with the two-step starting with the third configuration. Let us remark that considering different initial configurations in the outer loop allows moving towards better solutions. Other existing optimization heuristics [31] [22] consider only one random initial configuration.

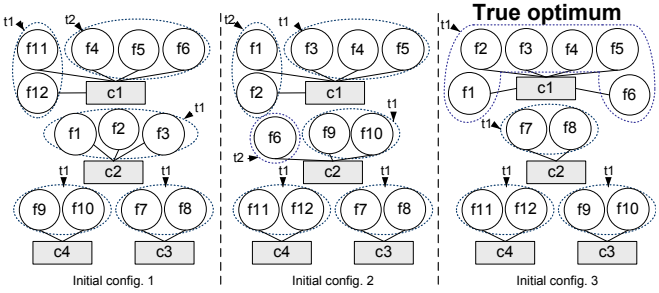


Figure 11. Solutions for all configurations - MILP

Table 2. Intermediate results for each initial configuration

Initial configs	Steps	Metrics	Iter 1	Iter 2	Iter 3	Iter 4	Iter 5
1	PP	O_s	5.93	7.45	7.45	7.45	7.45
		O_l	112.69	96.67	78.85	72.75	71.82
	PS	O_s	7.45	7.45	7.45	7.45	7.45
		O_l	103.09	81.85	76.23	71.82	71.82
2	PP	O_s	7.45	7.45	7.45	7.45	-
		O_l	89.47	74.32	71.82	-	-
	PS	O_s	7.45	7.45	7.45	7.45	-
		O_l	78.32	71.82	71.82	-	-
3	PP	O_s	7.45	-	-	-	-
		O_l	68.27	-	-	-	-
	PS	O_s	7.45	-	-	-	-
		O_l	68.27	-	-	-	-

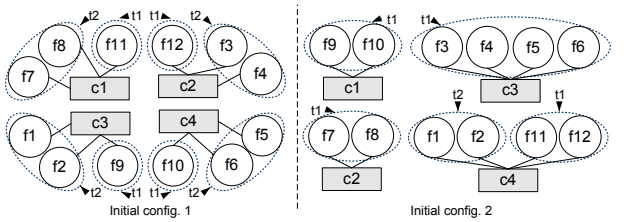


Figure 12. Solutions for configuration 1 and 2 - GA

5.2 Two-step techniques evaluation on an industrial-size system

The objective of the following experiment is to study the behavior of the two-step MILP and GA in the case of a large, distributed industrial-size system as in [6], but where the partitioning is not fixed. Figure 14 shows the functional graph of the system. The system has 9 nodes communicating through a single bus. The maximum capacity utilization is 1 for the bus and nodes. Nodes are heterogeneous and have different computation speeds. The WCETs and nodes allocation constraints for this system are given in Table 3. The WCTT for all signals is 10. In the initial configuration each function is partitioned in one task and the task priority is inversely proportional to the index of the function in it (i.e. the highest priority task contains the function with the smallest index).

The solution obtained with the two-step MILP for the first iteration is shown in Figure 13. The minimal slack for the two-step MILP is 2396 for the PP step and 3269 after the PS step (for the first iteration). The two-step GA computes slightly worse results, i.e. a minimum slack of 2387 for the PP step and 3262 for the PS step. Latency (in the first iteration) is the same for both methods, 9981 for the PP, and 6251 for the PS step.

5.3 Runtime and scalability evaluation

In this section we show respectively (i) runtime evaluation of all the techniques, pointing out the limits of one-step MILP when the size of the system grows and (ii) runtime evaluation of the two-step techniques for large systems, showing their scalability.

5.3.1 Runtime evaluation

In the first experiment, we are interested in showing the limits of the one-step MILP. We generate systems of 5,10,15 and 20 functions, by combining the transactions of Figure 7. The network topology is composed of four nodes communicating by two buses, the first bus connects nodes 1, 2 and 3 and the second nodes 3 and 4. The maximal capacity utilization is 1 for all nodes and buses. We run two outer loops for two-step techniques. We compare the quality

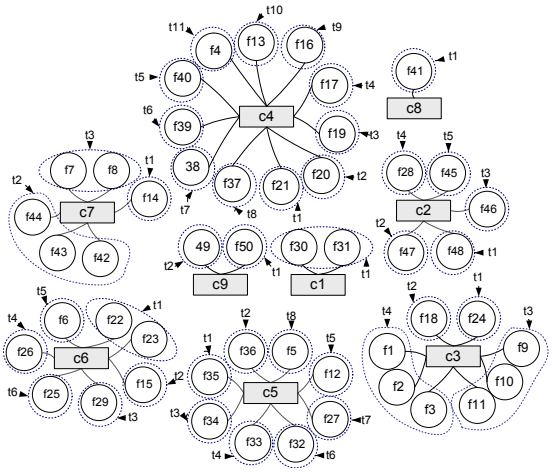


Figure 13. Deployment solution for the large case

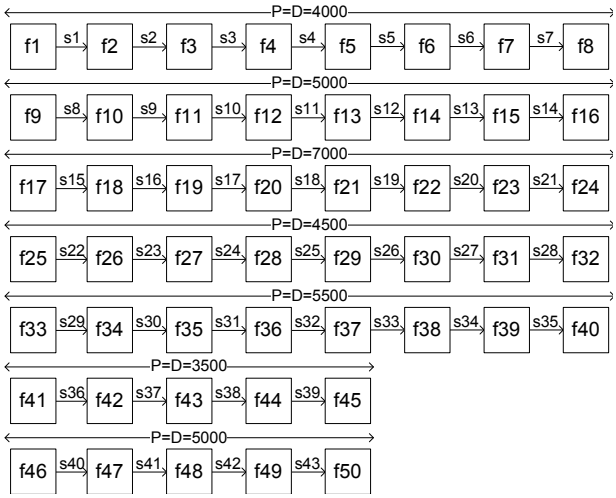


Figure 14. Functional graph of the large case

of solutions returned by one-step and two-step techniques, then we study the time required to compute the solution.

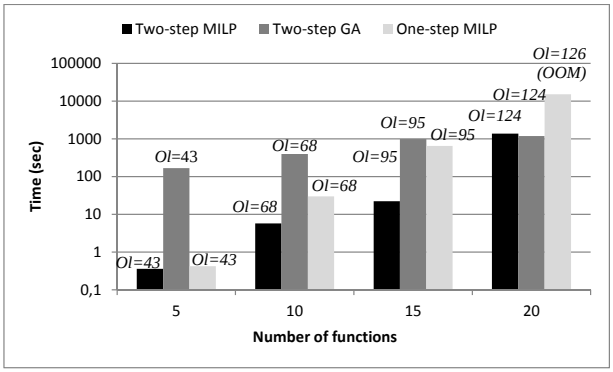


Figure 15. Performance of the two-step optimization technique vs the one-step approach

The results of the runtime comparison are shown in Figure 15: the runtime of all techniques increases according to the number of functions. The runtime of the two-step techniques (for two outer loops) is very small compared to the one-step MILP runtime, and the one-step MILP runs out of memory (OOM) for systems with 20 functions, while the two-step MILP computes a solution in 1375 seconds. Let us note that in all the cases the one-step MILP returns without error, the obtained values of latency are the same for all algorithms.

Table 3. Vectors of WCETs

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9		c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
f_1	48	26	35	50	36	40	45	27	28	f_{26}						48	36		
f_2	36	52	59	34	26	43	50	48	49	f_{27}			26		50				
f_3			49							f_{28}		51		30					
f_4				50						f_{29}						32	39		
f_5					46					f_{30}	50	48	36	26	35	26	35	50	51
f_6						50				f_{31}	31	39		44					
f_7							51			f_{32}			52		27				
f_8	26	35	29	42	50	37	20	52	53	f_{33}	44	26	35	26	35	30	28	49	50
f_9	50	40	31	29	33	46	37	29	30	f_{34}					30	40	50		
f_{10}			52							f_{35}					49	39	29		
f_{11}	40	39	50	33	36	39	43	41	42	f_{36}					53	25	40		
f_{12}					50					f_{37}		32	40	48					
f_{13}				39						f_{38}		30	50	45					
f_{14}							38			f_{39}		54	28	36					
f_{15}						52				f_{40}	50	28	39	47	44	35	26	35	36
f_{16}	33	46	37	29	35	29	42	50	51	f_{41}	52	28	35	26	35	50	45	33	34
f_{17}		32		55		29				f_{42}					48	38	28		
f_{18}			51		26					f_{43}					30	50	40		
f_{19}	51	42	26	35	50	36	33	29	30	f_{44}					26	38	48		
f_{20}		50		28			36			f_{45}		51	30	42					
f_{21}	35	29	42	42	26	35	50	54	55	f_{46}		34	49	30					
f_{22}						29	50			f_{47}	28	39	47	50	26	35	50	51	52
f_{23}						48	32			f_{48}		36	29	53					
f_{24}			31		45					f_{49}	51	44	33	28	50	26	39	33	34
f_{25}	53	26	35	50	36	26	35	50	51	f_{50}	37	50	52	32	29	27	36	40	41

5.3.2 Scalability evaluation of two-step techniques

To study the scalability of the two-step techniques, we generated systems of different sizes by combining the transactions shown in Figure 14. We consider only the inner loop iteration for two-step techniques (one initial configuration). The obtained results are shown in Table 4, where n/a means that there is no solution from the MILP solver, (*) that the solver runs out of memory, (#) that the a solution is obtained after the PP stage, but the solver runs out of memory in the PS stage. The table shows that the two-step MILP (2-MILP) is more scalable (up to 80 functions) than the one-step MILP (1-MILP, up to 40 functions), but the 2-GA is the most scalable approach.

Table 4. Results on systems with different sizes

	Maximal MinSlack (O_s)			Runtime (seconds)		
Nb functions	1-MILP	2-MILP	2-GA	1-MILP	2-MILP	2-GA
32	3500	3430	3490	10278	65.07	553.12
40	3388*	3462	3490	21533*	81.5	830.52
50	n/a	3269	3262	n/a	1744	1345
60	n/a	2347#	3131	n/a	732#	1862
80	n/a	926#	3180	n/a	1260#	2667.95
100	n/a	n/a	3017	n/a	n/a	5316.11
200	n/a	n/a	2039	n/a	n/a	16705.21

Please note that other approaches solving task placement and scheduling [6] [30] [25] do not deal with more than 50 tasks and 9 nodes.

6. Conclusions

We presented two approaches to optimize the deployment of (hard) real-time distributed systems w.r.t end-to-end latency metrics. We consider an event-triggered activation model and define the placement, partitioning and scheduling of functions and signals. We provided an MILP integral formulation for the problem, which can compute the optimal solution for small size systems and a dual formulation, based on MILP and Genetic algorithm (GA) for a partitioned, two-stage iterative approach. We evaluated the performance of the two-step approach by comparing the results against the optima for some small systems and then applied to larger case studies, including an automotive system. Future work includes an improvement of the MILP formulation for application to larger systems, an extension to more complex functional graphs, in which transactions are non-linear, and finally, the consideration of other optimization metrics.

References

[1] `http://www.omg.org/`.

[2] Autosar 4.0 specifications. `http://www.autosar.org/`.

[3] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, To appear.

[4] S. Anssi, S. Tucci-Piergiovanni, S. Kuntz, S. Gerard, and F. Terrier. Enabling scheduling analysis for autosar systems. *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, 0:152–159, 2011.

[5] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 127–132, 1991.

[6] E. Azketa, J. Uribe, J. Gutiérrez, M. Marcos, and L. Almeida. Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems. In *Proceedings of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 958–965, 2011.

[7] C. Bartolini, G. Lipari, and M. Di Natale. From functional blocks to the synthesis of the architectural model in embedded real-time applications. In *Proc. 11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 458–467, 2005.

[8] I. Bate and P. Emberson. Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 221–230, 2006.

[9] E. Bini, M. D. Natale, and G. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *Euromicro Conference on Real-Time Systems*, Dresden, Germany, June 2006.

[10] R. Bosch. CAN specification, version 2.0. Stuttgart, 1991.

[11] L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[12] R. Davis, A. Burns, R. Bril, and J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[13] A. Hamann, R. Racu, and R. Ernst. Multi-dimensional robustness optimization in heterogeneous distributed embedded systems. In *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium*, April 2007.

[14] X. He, Z. Gu, and Y. Zhu. Task allocation and optimization of distributed embedded systems with simulated annealing and geometric programming. *The Computer Journal*, 53(7):1071–1091, 2010.

[15] B. Kienhuis, E. Deprettere, P. Van Der Wolf, and K. Vissers. A methodology to design programmable embedded systems. In *Embedded processor design challenges*, pages 321–324. Springer, 2002.

[16] S. Kodase, S. Wang, and K. Shin. Transforming structural model to runtime model of embedded software with real-time constraints. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 170–175, 2003.

[17] S. Kugele, W. Haberl, M. Tautschnig, and M. Wechs. Optimizing automatic deployment using non-functional requirement annotations. *Leveraging Applications of Formal Methods, Verification and Validation*, pages 400–414, 2009.

[18] A. Mehiaoui. A mixed integer linear programming formulations for optimizing timing performance during the deployment phase in real-time systems design. *Technical Report CEA, http://hal-cea.archives-ouvertes.fr/cea-00811359*, 2012.

[19] B. L. Miller, B. L. Miller, D. E. Goldberg, and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[20] C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard. Optimum: a marte-based methodology for schedulability analysis at early design stages. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, 2011.

[21] J. C. Palencia and M. G. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, 1999.

[22] P. Pop, P. Eles, Z. Peng, and T. Pop. Analysis and optimization of distributed real-time embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 11(3):593–625, 2006.

[23] T. Pop, P. Eles, and Z. Peng. Design optimization of mixed time/event-triggered distributed embedded systems. In *Proc. of the First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2003.

[24] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 160–169, 2005.

[25] M. Richard, P. Richard, and F. Cottet. Allocating and scheduling tasks in multiple fieldbus real-time systems. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, 2003.

[26] M. Saksena, P. Karvelas, and Y. Wang. Automatic synthesis of multi-tasking implementations from real-time object-oriented models. In *Proceedings of 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 360–367, 2000.

[27] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design and Test of Computers*, 18(6):23–33, 2001.

[28] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2-3):117–134, 1994.

[29] W. Zheng, M. Di Natale, C. Pinello, P. Giusto, and A. Vincentelli. Synthesis of task and message activation models in real-time distributed automotive systems. In *Proceedings of the conference on Design, automation and test in Europe*, pages 93–98, 2007.

[30] Q. Zhu, Y. Yang, M. Di Natale, E. Scholte, and A. Sangiovanni-Vincentelli. Optimizing the software architecture for extensibility in hard real-time distributed systems. *IEEE Transactions on Industrial Informatics*, 6(4):621–636, 2010.

[31] Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, and A. Sangiovanni-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 275–284, 2009.

[32] Q. Zhu, H. Zeng, W. Zheng, M. Di Natale, and A. Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems*, 11(4):85:1–85:30, 2012.